COMP 302 Winter 2025 Problem Set 2

Problem 1: Factorial with Exception

Implement a function to compute factorial that raises an exception on negative input and explain why this function has type `int -> int`.

```
exception Negative_input

let rec factorial n =
 (* Implement *)
```

Problem 2: N-ary Tree Recursive Search

Implement a higher-order function to find a node that satisfies a predicate for an n-ary tree where each node has a list of subtrees.

```
exception NotFound

type 'a ntree = Empty | Node of 'a * 'a ntree list

let rec find p t = match t with
    (* Implement *)
```

Problem 3: Recursive Search with Embedded Helper Function

Perform a recursive search on a tree similar to Problem 2 but embed a helper function using `let-in` to manage the recursive logic.

Problem 4: Recursive Search with Fold

Perform a recursive search on a tree like in Problem 2, but fold list recursion with tree recursion to streamline the recursive process.

Problem 5: Coffee System

You are tasked with implementing a coffee system Each customer can open an account to track their coffee purchases. For every 5th coffee purchase, the coffee should be free. Otherwise, the coffee costs a fixed price.

Requirements

1. Implement a function to open a new coffee account. Each account tracks the number of coffees a customer has purchased.
2. Implement a function to simulate the purchase of coffee. The function should return the price of the coffee (e.g., 2 units per coffee, and free on every 5th purchase).
3. Implement a function that returns the total number of coffees purchased on an account.

Function Signatures

```
val open_coffee_account : unit -> coffee_account
val buy_coffee : unit -> int
val get_purchases : unit -> int
```

Example Usage and Expected Output

```
let my_account = open_coffee_account ();;

my_account.buy_coffee ();;  (* Returns 2 *)
my_account.buy_coffee ();;  (* Returns 2 *)
my_account.buy_coffee ();;  (* Returns 2 *)
my_account.buy_coffee ();;  (* Returns 2 *)
my_account.buy_coffee ();;  (* Returns 0, indicating free coffee *)
my_account.get_purchases ();;  (* Returns 5 *)
```

Problem 6: Simulating Mutable Lists

Simulate mutable lists using an immutable one.

```
type 'a mut_list = {
    append : 'a -> unit;
    drop : int -> unit;
    get_list : unit -> 'a list;
}

let make_mut_list l =
    let v = ref l in
    (* Implement *)
```

Example Usage and Expected Output

```
(* Creating a mutable list with initial elements *)
let myList = make_mut_list [1; 2; 3; 4; 5];;

(* Appending an element to the list *)
myList.append 6;;
myList.get_list ();;  (* Returns [1; 2; 3; 4; 5; 6] *)

(* Dropping the first element *)
myList.drop 2 ();;
myList.get_list ();;  (* Returns [3; 4; 5; 6] *)
```